Лабораторная работа №3. События и уведомления

Теоретическая часть

События и уведомления позволяют приёмнику события узнать об изменении состояния источника и среагировать на это изменение соответствующим образом. Приёмники события заранее подписываются на событие или уведомление источника. Может быть осуществлена подписка нескольких приёмников с одним источником, так и одного приёмника с несколькими источниками. Для реализации событий и уведомлений реагирующие на них обработчики делегируются. Делегирование может осуществляться явно с применением собственных интерфейсов либо неявно с использованием специальных интерфейсов и классов. Неявное делегирование применяется при обработке стандартных событий, таких как, например, события мыши и кнопок, существенно облегчая ихреализацию.

Явное событие и интерфейс

При явной реализации события удобно воспользоваться интерфейсом, поскольку использование интерфейсной ссылки позволяет ссылаться к разнотипным объектам (приемникам), классы которых наследуют один и тот же интерфейс, связанный с этим событием.

Класс источника должен иметь конструктор с параметром — интерфейсной ссылкой. Класс каждого приёмника этого события обязан наследовать этот интерфейс, включающий делегированный обработчик события.

Генерирование события источником сводится к вызову обработчика приёмника, ссылка на который передаётся в источник через интерфейсную ссылку в конструкторе.

Поясним вышесказанное.

```
/* Явное событие*/
interface IEv {void Handler();} // Интерфейс события
class Source {// Класс источника события
 IEv iEv;
 Source (IEv iEv) {this.iEv= iEv;} // Конструктор
 void genEv() {iEv.Handler();} // Генерировать событие
}
class Receiver implements IEv {// Класс приёмника события
 public void Handler() {System.out.println ("OK");}// Обработчик
}
class TestEvent {
 public static void main (String[] args) {
  Receiver receiver = new Receiver();// Создать объект приемника
  Source source = new Source(receiver);// Создать объект источника
 source.genEv();
 }
}
/*Result:
OK*/
```

Уведомления

Класс Observable и интерфейс Observer пакета java.util позволяют воспользоваться механизмом уведомления языка Java. Наблюдаемые, являющиеся объектами-источниками,

классы которых наследуют класс Observable, уведомляют обозревателей, являющимися объектами-приёмниками, классы которых осуществляют интерфейс Observer. Класс наблюдаемого объекта Observable переопределяет функции setChanged(), сигнализирующую об изменении состояния объекта, и функцию notifyObservers(), уведомляющую обозреватель и передающую ему объект с данными. Интерфейс Observer обязывает обозреватель переопределить функцию update(), реагирующую на уведомление и получающую объект данных.

Проиллюстрируем применение механизма уведомления.

```
/* Уведомление*/
import java.util.*;
class beWatched extends Observable {//Класс наблюдаемого объекта
 void notifyObs() {
  setChanged();
  notifyObservers(new Integer(55));
 }
}
class Watcher implements Observer {//Класс обозревателя
 public void update(Observable obs, Object arg) {
  System.out.println ("received " + ((Integer)arg).intValue());
 }
}
class TestEvent2 {
 public static void main (String[] args) {
  Watcher w = new Watcher();//Создать объект приемника
  beWatched bW = new beWatched();//Создать объект источника
  bW.addObserver(w);
  bW.notifyObs();//Уведомить
 }
}
/*Result: received 55*/
```

Применение уведомлений упрощает разработку программы, поскольку используемые класс Observable, интерфейс Observer и их функции стандартизируют разработку. Нет необходимости разрабатывать и применять свой интерфейс, строить особым образом класс источника и приёмника, как показано в примере применительно к явным событиям. К тому же класс Observable и интерфейс Observer включают иные полезные функции, например, такие как countObservers(), addObservers() deleteObservers() и другие. Единый подход к обработке событий, конечно, не только упрощает программирование, но и делает программы, написанные разными людьми, более понятными.

Обработка событий

При разработки оконных приложений как правило обрабатываются события, связанные как с окном, так и с консолью, мышью и интерфейсными элементами. Пакет java.awt.event включает классы ActionEvent, AdjustmentEvent, ComponentEvent, FocusEvent, InpunntEvent, KeyEvent, MouseEvent, TextEvent и WindowEvent источников событий, наследующих суперкласс EventObject, и интерфейсы ActionListener, AdjustmentListener, ComponentListener, ContainerListener, FocusListener, InputListener, KeyListener, MouseListener, MouseMotionListener, TextListener и WindowListener приёмников событий (интерфейсы прослушивания событий). Основываясь на принципах построения и

применения этих событий можно создать собственное событие и реализовать его обработку.

Каждый класс источника содержит одноимённую функцию подписки (регистрации) приёмников (блоков прослушивания) этого события формата

public void addTypeListener (TypeListener tL),

где Type – имя события, а tL – ссылка на объект приёмника события (ссылка на блок прослушивания, анонимный класс).

Класс, объект которого должен генерировать событие, обязан:
1) осуществлять (implements) интерфейс, соответствующий этому событию,
2) подписать на это событие требуемый приёмник (блок прослушивания).
Приёмник (блок прослушивания) должен:
1) переопределить предопределённый обработчик события.

В зависимости от события предопределённым обработчиком может быть actionPerformed() или другие.

Пример 1. Программа, где используется ввод с консоли

```
/* Пример ввод с консоли */
import java.util.*;
public class JavaApplication13 {
 public static void main(String[] args) {
  Scanner sc = new Scanner(System.in);
  int i = 0;
  System.out.print("Введите целое число: ");
// возвращает истинну если с потока ввода можно считать целое число
  if(sc.hasNextInt()) {
/*Аналогично метод hasNextDouble(), применённый объекту класса Scanner, проверяет, можно ли считать с потока ввода вещественное число типа double, а метод nextDouble() — считывает его.*/
// считывает целое число с потока ввода и сохраняем в переменную
  i = sc.nextInt();
  System.out.println(i*2);
  } else {
   System.out.println("См. выше ^^^");
  }
/*Метод nextLine(), позволяющий считывать целую последовательность символов, т.е. строку. Далее создаётся два таких объекта, потом в них поочерёдно записывается ввод пользователя, на экран выводится одна строка, полученная объединением введённых последовательностей символов.
*/
  Scanner sc2 = new Scanner(System.in);
  String s1, s2;
  s1 = sc2.nextLine();
  s2 = sc2.nextLine();
  System.out.println(s1 + s2);
 }
}
```

Пример 2. Программа, где используется ввод/вывод в файл

```
/* Пример работы с текстовым файлом */
import java.io.*;
public class JavaApplication12 {
 public static void main(String[] args) {
//Создание файла и запись в него ----------------------------------------
 String filename = "log.txt";
 File f = new File(filename);
 try{
  if(!f.exists()) f.createNewFile();
   PrintWriter pw = new PrintWriter(f.getAbsoluteFile());
  try{
   pw.println("2 ~!");
   pw.println("4 ~!");
   pw.println("6 ~!");
   pw.println("8 ~!");
  }finally{pw.close();}
 }catch(IOException e){throw new RuntimeException();}
 //Чтение из файла ------------------------------------------------------
 StringBuilder sb = new StringBuilder();
 if(f.exists()){
  try{
   BufferedReader br = new BufferedReader(new FileReader(f.getAbsoluteFile()));
   try{
    String s;
    while((s = br.readLine())!=null){//построчное чтение
     sb.append(s);
     sb.append("\n");
    }
   }finally{br.close();}
  }catch(IOException e){throw new RuntimeException();}
 }
 System.out.print(sb.toString());//в sb.toString() хранится текст файла
/*Для обновления файла можно сперва прочитать файл в переменную типа String,
как это сделано в секции чтение из файла,
а затем, сложив его с новым содержимым, записать в файл*/
//Удаление файла ---------------------------------------------------------
//f.delete();
 }
}
```

Практическая часть

1. Реализовать ввод данных с консоли и из указанного файла (например, в задании нужно задать число N, то можно поступить так: с консоли вы вводите путь к фалу, где записано значение числа N и возможно что-то ещё см. П.2).
2. Все операции вывода на консоль должны дублироваться выводом в указанный файл «Журнала» (путь к нему задать либо с консоли − «0», либо из файла – «1», который упомянут в П.1). Вариант задания файла «Журнала» смотри в таблице 2.
3. При написании необходимого класса следует пользоваться одним из двух способов генерации, перехвата и обработки событий, а именно: явная реализация события – «0» и использование класса Observable и интерфейса Observer – «1». Тип способа осуществления события в разрабатываемом классе указан в таблице 2.
4. Показ использования функций класса обеспечить созданием объектов класса в функции public static void main(String[] args) и вызовом функций этих объектов. При показе следует в череде примеров предъявлять факты генерации, перехвата и обработки требуемых событий. Включить в отчёт исходные данные тестов использования функций класса и соответствующие им результаты вывода на консоль и в файл «Журнала».

Варианты заданий

Выполнить задание из первой лабораторной работы, но, чтобы при этом в описании требуемого класса учитывались (генерировались, перехватывались и обрабатывались) указанные ниже события (см. таблицу №2).

Таблица №1

№№	Событие
1.	Обращение к потоку вывода на консоль
2.	Обращение к указанному массиву
3.	Равенство указанного объекта некоторому значению
4.	Обращение к потоку ввода с консоли
5.	Изменение указанной переменной
6.	Обращение к потоку вывода в указанный файл
7.	Получение некоторого результата работы функции
8.	В массиве число элементов равно указанному
9.	Обращение к потоку ввода из указанного файла

Номер отличного от нуля разряда в таблице №2 соответствует номеру события в таблице №1, которое должно быть осуществлено в разрабатываемом классе. Так, например, вариант №1 из таблицы №2 соответствует следующим событиям: «Обращение к потоку вывода на консоль», «Обращение к указанному массиву» и «Равенство указанного объекта некоторому значению».

Таблица №2

№	Номера событий	Указание пути к фалу «журнала»	Способ реализации событий
1.	000000111	0	1
2.	000001011	1	0
3.	000001101	0	1
4.	000001110	1	0
5.	000010011	0	1
6.	000010101	1	0
7.	000010110	0	1
8.	000011001	1	0
9.	000011010	0	1
10.	000011100	1	0
11.	000100011	0	1
12.	000100101	1	0
13.	000100110	0	1
14.	000101001	1	0
15.	000101010	0	1
16.	000101100	1	0
17.	000110001	0	1
18.	000110010	1	0
19.	000110100	0	1
20.	000111000	1	0
21.	001000011	0	1
22.	001000101	1	0
23.	001000110	0	1
24.	001001001	1	0
25.	001001010	0	1
26.	001001100	1	0

27.	001010001	0	1
28.	001010010	1	0
29.	001010100	0	1
30.	001011000	1	0
31.	001100001	0	1
32.	001100010	1	0
33.	001100100	0	1
34.	001101000	1	0
35.	001110000	0	1
36.	010000011	1	0
37.	010000101	0	1
38.	010000110	1	0
39.	010001001	0	1
40.	010001010	1	0
41.	010001100	0	1
42.	010010001	1	0
43.	010010010	0	1
44.	010010100	1	0
45.	010011000	0	1
46.	010100001	1	0
47.	010100010	0	1
48.	010100100	1	0
49.	010101000	0	1
50.	010110000	1	0
51.	011000001	0	1
52.	011000010	1	0
53.	011000100	0	1
54.	011001000	1	0
55.	011010000	0	1
56.	011100000	1	0
57.	100000011	0	1
58.	100000101	1	0
59.	100000110	0	1
60.	100001001	1	0
61.	100001010	0	1
62.	100001100	1	0
63.	100010001	0	1
64.	100010010	1	0
65.	100010100	0	1
66.	100011000	1	0
67.	100100001	0	1
68.	100100010	1	0
69.	100100100	0	1
70.	100101000	1	0
71.	100110000	0	1
72.	101000001	1	0
73.	101000010	0	1
74.	101000100	1	0
75.	101001000	0	1
76.	101010000	1	0

77.	101100000	0	1
78.	110000001	1	0
79.	110000010	0	1
80.	110000100	1	0
81.	110001000	0	1
82.	110010000	1	0
83.	110100000	0	1
84.	111000000	1	0